Chapter 9
On the Early History and Impact of Unix Tools to Build the Tools for a New Millennium

"When the barbarian, advancing step by step, had discovered the native metals, and learned to melt them in the crucible and to cast them in moulds; when he had alloyed native copper with tin and produced bronze; and, finally, when by a still greater effort of thought he had invented the furnace, and produced iron from the ore, nine tenths of the battle for civilization was gained. Furnished, with iron tools capable of holding both an edge and a point, mankind were certain of attaining to civilization."

Lewis Henry Morgan

"When Unix evolved within Bell Laboratories, it was not a result of some deliberate management initiative. It spread through channels of technical need and technical contact.... This was typical of the way Unix spread around Bell Laboratories.... I brought out that little hunk of history to point out that the spread and success of Unix, first in the Bell organizations and then in the rest of the world, was due to the fact that it was used, modified, and tinkered up in a whole variety of organizations."

Victor Vyssotsky

"UNIX is a lever for the intellect."

John R. Mashey

Iron Tools and Software Tools

Our era is witnessing the birth of an important new technology, different from any in the past. This new technology is the technology of software production. The new tools of our era are tools that make it possible to produce software. Unlike the tools forged in the past, software tools are not something you can grab or hold. Similarly the processes required to develop this new technology are new. Knowing the details of how such new tools have been created will make it possible to better understand the promise they present for our times and for the future.

Describing a previous technological leap, the American anthropologist Lewis Henry Morgan called the production of iron "the event of events in human experience…. Out of it," he wrote, "came the metallic hammer and anvil, the axe and the chisel, the plow with an iron point, the iron sword; in fine, the basis of civilization which may be said to rest upon this metal." Morgan maintained that until the human species had mastered the skills and methods necessary for forging iron, social development had been paused awaiting the achievement of this next necessary technological advance. "The want of tools," he wrote, "arrested the progress of mankind in barbarism. There they would have remained to the present hour, had they failed to bridge the chasm." Frustrated by the lack of knowledge about how iron was invented, Morgan lamented, "It would be a singular satisfaction could it be known to what tribe and family we are indebted for this knowledge and with it for civilization."[1]

We are more fortunate than our ancestors. We can know some of the details of whom to credit for the technological achievements of the computer revolution. Documenting their contributions while many of the pioneers are still alive will provide future generations with knowledge of whom they are indebted to for this new and stunning advance. Like the invention of iron, the technology of software production promises to make possible an important new level of civilization. Such knowledge will also help those who want to build on what has already been achieved.

Creating a Technology of Software Development

In Introduction to Computer Science, Alan Perlis, one of the early pioneers of computer science, observed that the nature of computer programming is too often misunderstood. "Although computer programming is often inaccurately considered an art," he wrote, "it is actually an infant technology."[2]

Like Morgan, Perlis believed that the progressive and accelerating ability to use and produce tools represents a defining element in the advance of human society. He observed that the rapid increase in the ability of the human species to use and mold new tools was guided by two laws. The first law describes the rapid increase in the number of people who can wield the tools and in the number of tools available. The second describes how the rapid development of tools to create tools guarantees the continuing expansion of the ability to use the tools. Perlis wrote: "First, the increase in the population trained to use tools and in the number of tools available for their use implies that the potential for tool utility grows as a product rather than as a sum. Second, the amount of composition – that is the number and uses of tools to produce new tools – grows as a part of the product and assures an even greater overall growth in tool use."[3]

Among the tools Perlis saw coming into common use "are those which perform automatically at great speed, and whose management must perform in the same way. Here," he explained, "the computer is essential because of the need for quick response."[4]

The Creation of Time-Sharing

The development of the modern computer raised the question of how to manage its operation more efficiently. An important solution to the problem of computer management proposed in 1959 was the sequential and automatic interspersing of many tasks known as time-sharing.[5] Among the earliest time-sharing systems were those developed at MIT. By 1963, there were two versions of the Compatible Time Sharing System (CTSS) operating at MIT on two IBM 7094 computers, one at the Computation Center, and another at MIT's Project MAC.

Those using the early time-sharing systems at MIT and elsewhere soon discovered the delights of interactive computing made possible by time-sharing.[6] Describing the advantages of interactive computing, time-sharing pioneers Robert Fano and Fernando Corbato, wrote: "For professional programmers the time-sharing system has come to mean a great deal more than mere ease of access to the computer. Provided with the opportunity to run a program in continuous dialogue with the machine, editing, 'debugging' and modifying the program as they proceed, they have gained immeasurably in the ability to experiment. They can readily investigate new programming techniques and new approaches to problems."[7]

The results of this programming flexibility led both to a bolder and more flexible approach to problem solving and to undertaking new areas of research. Fano and Corbato reported that users not only would build on each other's work, but also they would come to depend more and more on the computer to facilitate their work. The most surprising development that they encountered, however, was the fact that the users themselves created many of the programming commands used in the system, instead of needing professional programmers. While at the conventional computer installation, they noted, "one hardly ever makes use of a program developed by another user, because of the difficulty of exchanging programs and data," in the Project MAC time-sharing environment, "the ease of exchange has encouraged investigators to design their programs with an eye to possible use by other people. They have acted essentially as if they were writing papers to be published in technical journals."[8]

Fano and Corbato envisioned that time-sharing systems would have a profound impact on the future. "Communities will design systems," they predicted, "to perform various functions – intellectual, economic and social – and the systems in turn undoubtedly will have profound effects in shaping the patterns of human life."[9] "The coupling between such a utility and the community it serves," they discovered, "is so strong that the community is actually a part of the system itself." They foresaw the development of a symbiotic relationship between the computer systems and its human users which "will create new services, new institutions, a new environment and new problems." Among these, they proposed, would be the question of access. "How will access to the utility be controlled?" they asked, "To what ends will the system be devoted, and what safeguards can be devised for its misuses? It is easy to see," they concluded, "that the progress of this new technique will raise many social questions as well as technical ones."[10]

Others during this period were concerned with the impact the computer would have on current society. For example, John McCarthy predicted that, "The computer gives signs of becoming the contemporary counterpart of the steam engine that brought on the industrial revolution."[11] Unlike the steam engine, however, the utility of the computer was dependent on the successful development of software programs written to direct it. Therefore, along with the increasing speed and capacity of computer hardware, came the increase in the demand for and in the cost of software. By the mid 1960's, the U.S. government was spending increasing amounts of money to create programs to utilize computer resources. The U.S. government, wrote McCarthy, "with a dozen or so big systems serving its military and space establishments, is spending more than half of its 1966 outlay of $844 million on software."[12]

Pointing out the need for studying the processes of programming, McCarthy observed, "What computers can do, depends on the state of the art and the science of programming as well as on speed and memory capacity."[13] Computer pioneers like McCarthy recognized that the computer was more than an efficient bookkeeping machine. There was a need to discover what new applications were possible, and to create these new applications. Therefore, there would be a need for breakthroughs in the process of programming software. McCarthy believed that it was important for the user to be able to program in order to realize the potential of the computer. He pointed out that programming was a skill that was not difficult to learn, and that it was more important to understand the task being automated than to master programming languages. "To program the trajectory of a rocket," McCarthy offers as an example, "requires a few weeks' study of programming and a few years' study of physics."[14]

These early explorations in time-sharing prepared the foundation for an important development in the process of creating software. Once the discovery was made that simple programming tools could be created to aid in the process of software development, and that such tools could help those who understood the tasks to be automated, a needed leap could be made in how to develop software. Such a program was to be carried out by research programmers and developers at Bell Labs in the 1970's and early 1980's, building on the principles developed by the pioneers of time-sharing and Project MAC.

The Multics Collaboration

In 1964, MIT joined with GE and AT&T in a project designed to implement time-sharing by developing a new computer and a new operating system. The joint research project among GE, MIT, and AT&T was created to extend time-sharing techniques from a pilot program into a useful prototype for the future information utility.[15] The researchers realized that there was no existing computer that would meet the demands of time-sharing. Therefore part of the goal of their collaboration was to make it possible to develop a new computer as well as a new operating system.

The collaborative project was called Multics [Multiplexed Information and Computing Service] and was to be implemented on the GE 645 computer.[16] Technical leadership of the project included F. J. Corbato from MIT and V.A. Vyssotsky from Bell Labs. "One of the overall design goals is to create a computing system," they wrote, "which is capable of meeting almost all of the present and near-future requirements of a large computer utility. Such systems, must run continuously and reliably 7 days a week, 24 hours a day in a way similar to telephone or power systems, and must be capable of meeting wide service demands: from multiple man-machine interaction to the sequential processing of absentee-user jobs…."[17]

The goal of the research was to produce a prototype time-sharing system. Berkley Tague, one of the Bell Labs researchers involved in the Multics project writes, "The Multics Project was a joint project of Bell Labs, the GE Computer Systems Division, and MIT's Project MAC to develop a new computer and operating system that would replace MIT's CTSS system, Bell Labs BESYS, and support the new GE machine."[18] Though AT&T withdrew from the project in 1969, the joint work achieved significant results. Summarizing these achievements, Tague writes, "Multics was one of the seminal efforts in computing science and operating system design. It established principles and features of operating system design that are taken for granted today in any modern operating system."[19]

The Need for Operating System Research at Bell Labs

Even though AT&T withdrew from the research collaboration on Multics, computer scientists at Bell Labs wanted some way of continuing the advanced form of programming research that their work with CTSS and Multics had made possible. As early as 1957, Bell Labs had found they needed an operating system for their in-house computer center which was then running lots of short batch jobs. Describing the situation then facing the Labs, Vyssotsky explains, "We just couldn't take the time to get them on and off the machine manually. We needed an operating system to sequence jobs through and control machine resources."[20] The BESYS operating system was created at Bell Labs to deal with their in-house needs. When asked by others outside the Labs to make a copy available, they did so but with no obligation to provide support. "There was no support when we shipped a BESYS tape to somebody," Vyssotsky recalls, "we would answer reasonable questions over the telephone. If they found troubles or we found troubles, we would provide fixes."[21]

By 1964, however, Bell Labs was adopting third generation computer equipment and had to decide whether they would build their own operating system or go with one that was built outside the Labs. Vyssotsky recounts the process of deliberation at the time, "Through a rather murky process of internal deliberation we decided to join forces with General Electric and MIT to create Multics." He explains that the Labs planned to use the Multics operating system, "as a mainstay for Bell Laboratories internal service computing in precisely the same way that we had used the BESYS operating system."[22]

Unix Is Born and the Introduction of Pipes

When AT&T made the decision to pull out of the Multics collaboration, they took the research operating system off their GE 645 computer and put up the GECOS operating system. Though GECOS was adequate for applications, it was "nowhere near as satisfactory if you were trying to do things that were technically difficult and imperfectly defined," explained Vyssotsky, "which is the main task of research."[23]

For the pioneering work of Bell Labs research programmers like Ken Thompson and the research purposes of the Labs, an operating system more like what Multics had promised was needed. Along with the advantages of immediate feedback which time-sharing provided, the Bell Labs researchers wanted to continue the ability to work collaboratively which time-sharing had made possible.

"What we wanted to preserve," one of the creators of Unix, Dennis Ritchie writes, "was not just a good programming environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication."[24]

Ritchie describes how an informal group led by Thompson had begun investigating alternatives to Multics before the GE-645 Multics machine had been removed from the Labs.[25] Thompson and Ritchie presented Bell Labs with proposals to buy them a computer so they could build their own interactive, time-sharing operating system. Their proposals weren't acted on. Eventually, Thompson found a little used PDP-7 computer. According to Vyssotsky, the orphaned PDP-7 computer was a machine, "more in the class of a Commodore 64 than the class of a PC-AT."[26]

Ritchie, Thompson, and Rudd Canaday, who had been part of the Multics project, applied the lessons they had learned to the design of a file system for an experimental operating system. Writing on a chalk board, they created a file system design based on "the particularly simple way of viewing files that was suggested by the Multics I/O system."[27]

"Soon," Ritchie recounts, "Thompson began implementing the paper file system (perhaps 'chalk file system' would be more accurate) that had been designed earlier." Thompson was eager to get a working model so he could test it out. He proceeded to create the other aspects of an operating system. "A file system without a way to exercise it was a sterile proposition," notes Ritchie, "so he [Thompson] proceeded to flesh it out with the other requirements for a working operating system, in particular the notion of processes."[28]

Describing the primitive conditions that Thompson faced, Ritchie writes, "At the start, Thompson did not even program on the PDP itself, but instead used a set of macros for the GEMAP assembler on a GE-635 machine."[29] A paper tape was generated on the GE 635 and then tested on the PDP-7 until, according to Ritchie, "a primitive Unix kernel, an editor, an assembler, a simple shell (command interpreter), and a few utilities (like the Unix rm, cat, cp commands) were completed. At this point, the operating system was self-supporting, programs could be written and tested without resort to paper tape, and development continued on the PDP-7 itself."[30]

Ritchie describes how Thompson's PDP-7 assembler was a model of simplicity. "There were no libraries, no loader or link editor," he writes, "the entire source of a program was presented to the assembler, and the output file – with a fixed name – that emerged was directly executable."[31] Once the assembler was completed, "the system was able to support itself. And thus the operating system we now call UNIX was born," notes Ritchie.[32]

Among the other active contributors during this period were Bell Labs researchers Rudd Canaday and Joe Ossanna. The researchers were anxious to continue their work on a more advanced computer than the PDP-7. However, their efforts to get AT&T to buy them a more advanced computer for their time-sharing research hadn't succeeded. With the help of Ossanna and another Bell Labs researcher Lee McMahon, they were finally able to convince management to buy them a new PDP-11 computer. To obtain this agreement, though, the researchers promised to create a text processing system. Doug McIlroy explains that "typists everywhere were seen as potential customers of the promised document-preparation program. Only later did Ossanna spot the patent department as a ripe candidate. By then there had already been clients in the telephone-operations business."[33] By Spring 1971, the Bell Labs Unix pioneers had a text formatter, fulfilling their promise. They had translated one they had created for the PDP-7 from a program McIlroy had written in the BCPL language while working on Multics. The text formatter was an assembler program for the PDP-11.[34]

One of the important developments in Unix was the introduction of pipes. Pipes had been suggested by McIlroy during the early days of creating Unix. Ritchie explains how “the idea, explained one afternoon on a blackboard, intrigued us but failed to ignite any immediate action. There were several objections to the idea as put…. What a failure of imagination,” he admits.[35] McIlroy concurs, describing how the initial effort to add pipes to Unix occurred about the same time in 1969 that Ritchie, Thompson and Canaday were outlining ideas for a file system. “That was when,” he writes, “the simple pipeline as a way to combine programs, with data notationally propagating along a chain of (not necessarily concurrent) filters was articulated.”[36] However, pipes weren’t implemented in Unix until 1972. “Thanks to McIlroy’s persistence,” Ritchie writes, “pipes were finally installed in the operating system (a relatively simple job), and a new notation was introduced.”[37] Several of the old commands had to be changed to make them usable with pipes, notes Ritchie. Summarizing how pipes found their way into Unix, Vyssotsky notes that Thompson put them in, but “it was McIlroy who said, ‘look you ought to do it.’ Pipes, like most things in Unix were not a radically new idea.”[38] He describes how similar ideas had appeared in other languages like SIMULA as early as 1967.

Dick Haight, a Unix pioneer who helped develop the Programmer’s Workbench of Unix tools, was present the day that pipes were introduced into Unix. He describes what happened: “I happened to have been visiting the research crew the day they implemented pipes. It was clear to everyone practically minutes after the system came up with pipes working that it was a wonderful thing. Nobody would ever go back and give that up if they could help it.”[39] Also describing the day pipes were introduced, McIlroy writes: “Open Systems! Our Systems! How well those who were there remember the pipe-festooned garret where Unix took form. The excitement of creation drew people to work there amidst the whine of the computer’s cooling fans, even though almost the same computer access, could be had from one’s office or from home. Those raw quarters saw a procession of memorable events. The advent of software pipes precipitated a day-long orgy of one-liners…. As people reveled in the power of functional composition in the large, which is even today unavailable to users of other systems.”[40]

The Software Tool

Pipes had been created by the time the Version 3 Unix Manual appeared in February 1973. The date listed for the creation of pipes is January 15, 1973.[41] Not only were pipes a significant addition to Unix, but according to McIlroy, pipes made possible a subsequent important discovery. “In another memorable event,” he writes, “the unarticulated notion of software tools, which had been bolstered by pipes, was finally brought home by the liberation of the pattern matching program grep from within the editor.”

McIlroy describes how he asked Thompson to create a program to help him with some work that he was trying to do. This program resulted in the invention of the software tool ‘grep.’ Following is McIlroy’s account of how ‘grep’ was taken out from the editor, leading to a clearer understanding of the notion of a software tool. He writes: “Grep was invented for me. I was making a program to read text aloud through a voice synthesizer. As I invented phonetic rules I would check Webster’s dictionary for words on which they might fail. For example, how do you cope with the digraph ‘ui’, which is pronounced many different ways: ‘fruit’, ‘guile’, ‘guilty’, ‘anguish’, ‘intuit’, ‘beguine’? I would break the dictionary up into pieces that fit in ed’s limited buffer and use a global command to select a list. I would whittle this list down by repeated scannings with ‘ed’ to see how each proposed rule worked.”

“The process was tedious, and terribly wasteful, since the dictionary had to be split (one couldn’t afford to leave a split copy on line). Then ‘ed’ copied each part into /tmp, scanned it twice to accomplish the ‘g’ command, and finally threw it away, which takes time too.”

“One afternoon I asked Ken Thompson if he could lift the regular expression recognizer out of the editor and make a one-pass program to do it. He said yes. The next morning I found a note in my mail announcing a program named grep. It worked like a charm. When asked what that funny name meant, Ken said it was obvious. It stood for the editor command that it simulated, g/re/p (global regular expression print).”

“Progress on my talking program accelerated dramatically. From that special-purpose beginning, grep soon became a household word. (Something I had to stop myself from writing in the first paragraph above shows how firmly naturalized the idea now is: ‘I used ed to grep out words from the dictionary.’) More than any other single program, grep focused the viewpoint that Kernighan and Plauger christened and formalized in ‘Software Tools’: make programs that do one thing and do it well, with as few preconceptions about input syntax as possible.”[42]

Grep is listed in the Manual for Version 4 Unix which is dated November, 1973. The date given for the creation of grep is March 3, 1973, following the creation of pipes.[43] The creation of grep, McIlroy explains, was followed by the invention of other special purpose software programs that could be used as tools. He writes: “A while later a demand arose for another special-purpose program, gres, for substitution: g/re/s. Lee McMahon undertook to write it, and soon foresaw that there would be no end to the family: g/re/d, g/re/a, etc. As his concept developed it became sed, a tool of remarkable utility that is largely unappreciated today, because it capitalizes on the perfect familiarity with ed that was universal ten years ago, but no more. Sed covers a lot of needs. For example, we have never seen fit to adopt the widespread ‘head’ utility because ‘sed 10q’ does just as well.”[44]

What McIlroy refers to as “the unarticulated notion of software tools…brought home by the liberation of…grep from within the editor” has become one of the significant achievements of the Bell Labs research work on Unix. By making it possible to use ‘grep’ to search for a data pattern in a file, without having to use an editor to go inside the file, the Bell Labs researchers discovered that they could create a plethora of software tools to be used in varying combinations, thus facilitating the customized application by a user, a goal sought by those doing research in programming.[45]

McIlroy explains that the notion of "software tool" only became articulated among the Bell Labs researchers with the publication of the book "Software Tools" by Kernighan and Plauger. "Still unnamed in our circles," McIlroy notes, "until Kernighan and Plauger wrote their book, the idea nevertheless became a guiding principle."[46] McIlroy adds that "We were definitely building tools before K&P, though we lacked the suggestive word."[47]

Describing how the notion of software tools helps to create an effective programming environment, Brian W. Kernighan and Rob Pike, authors of The Unix Programming Environment, explain that each tool is designed to be used with other tools and achieving this end is more important than how each is designed internally.[48] The most important aspect considered with each tool is the function that the tool is to have. New tools are designed once a new function is intended.

"The guiding principle," they write, "for making the choice should be that each program does one thing. Options are appropriately added to a program that already has the right functionality. If there is no such program then a new program is called for. In that case the usual criteria for program design should be used: the program should be as general as possible, its default behavior should match the most common usage and it should cooperate with other programs."[49]

Thus Unix, according to Kernighan and Pike, created "a new style of computing, a new way of thinking of how to attack a problem with a computer. This style," they explain, "was based on the use of tools: using programs separately or in combination to get a job done, rather than doing it by hand, by monolithic self-sufficient subsystems, or by special-purpose, one-time programs."[50]

The philosophy of using software tools that developed from research in Unix is outlined in the "Foreword" to the special issue of The Bell System Technical Journal published in 1978, on "The Unix Time-sharing System." Describing the principles which they have found to be an important component of the Unix philosophy of software design, the researchers write: "UNIX utilities are usually thought of as tools – sharply honed programs that help with generic data processing tasks. Tools were often invented to help with the development of UNIX programs and were continually improved by much trial, error, discussion, and redesign, as was the operating system itself. Tools may be used in combination to perform or construct specific applications."[51]

They explain that a distinctive style evolved as part of Unix research. "Unix software works smoothly together; elaborate computing tasks are typically composed from loosely coupled small parts, often software tools taken off the shelf."[52]

"Sophisticated tools to make tools have evolved," they observe.[53] Software development tools such as "nroff" and "troff" were created. Not only was it important to create tools, but soon tools to create tools, like "yacc" and "lex", were developed. "Yacc" and "lex" were used to create numerous little languages and applications like eqn and awk that greatly enhanced the popularity of Unix.

The evolution of the tool "diff" created by McIlroy, a commonly used software tool, is an example of how tools were continually improved, based on the experience gained using them.

McIlroy reports how he based his work on algorithms created by others and then he tried three different algorithms before settling on the one finally used.[54] Other Unix tools were created through a similar process. Each program was created to fulfill some simple capability and was called a tool. The programs were designed to be fun to use and to be helpful to programmers. Among the principles guiding the tool-builders were:

> "(i) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
>
> (ii) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
>
> (iii) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
>
> (iv) Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them."[55]

"Our goals throughout the effort," write Ritchie and Thompson, in describing the research objectives of their work on Unix, "when articulated at all, have always been to build a comfortable relationship with the machine and to explain ideas and inventions in operating systems and other software."[56]

Frederick P. Brooks Jr. is another computer pioneer who recognized the importance of pipes and combining single function programs into what he calls "a unified programming environment" to make the work of programmers more productive. Brooks describes Unix as one of "the first integrated environments to come into widespread use," which "improved productivity by integral factors."[57]

In explaining how such an environment functions, he writes: "They attack the accidental difficulties that result from using individual programs together, by providing integrated libraries, unified file formats, and pipes and filters. As a result, conceptual structures that in principle could always call, feed, and use one another can easily do so in practice."[58]

Following the development of single function software tools which could be used together via pipes in a programming environment, came the development of whole tool-chests and workbenches of tools to rationalize the work of programmers.

Open Source Code and the Early Development of Unix

Since the Unix programming environment was used by the researchers themselves, they came to learn its weaknesses and problems and were encouraged to make needed improvements. Contributing to the value of Unix during its early development, was the fact that the source code was open and available. It could be examined, improved, and customized. In an article "Can UNIX Survive Secret Source Code?", Mike Lesk, then a Bell Labs researcher, observes how only when computer source code is open and can be modified, will it be developed and vibrant. He gives the example of COMIT, the string processing language. At one point, its owners decided there would no longer be any modifications in the code, and so only distributed it in binary form. "You haven't heard of COMIT since," he notes. He describes how the same fate befell TRAC, a language close to FORTH. "Software is more attractive to hackers," Lesk maintains, "if it can be changed. As more and more UNIX suppliers restrict access to source code, fewer and fewer advanced research shops will be attracted to UNIX."[59]

Commenting on the importance of open source code in the early years of Unix development at Bell Labs, Thompson and Ritchie write, "Because all source programs were always available and easily modified on-line we were willing to revise and rewrite the system and its software when new ideas were invented, discovered, or suggested by others."[60]

Not only was the source code open and available to the Bell Labs researchers developing the system, but the Labs also made the sources available on tape at the request of academic colleagues. Robert Fabry of the University of California at Berkeley was able to get a tape of Version 4 Unix and that began the long and important role played by faculty and students at the University of California at Berkeley in the development of Unix.[61]

Source code tapes were made available to interested researchers in the academic and computer science community for a nominal fee. For example, when John Lions, a Professor of Computer Science at the University of New South Wales in Australia read a paper published by Thompson and Ritchie in mid 1974, he wrote them for a copy of the Unix tape. After signing a license agreement with the Labs, and a token payment of $110 Australian ($150 U.S.), the Unix Edition 5 tape and manuals arrived, "as a late Christmas present," in December, 1974, remembers John Lions.[62]

While Bell Labs made the tape and manuals available, they did so with no support. Berkley Tague explains the release was "'caveat emptor' – i.e. dollars on the table up front and no support promised."

Henry Spencer, a Unix pioneer from the University of Toronto in Canada, and one of the programmers of C News, describes how early users of Unix in the academic community had to provide their own support. He explains: "It was very common at the time. This was in the days when UNIX was still treated by the Bell System as, 'Oh, just something we happen to do for our own internal use. You can have a copy if you want, but if you got problems, don't bother us.' And the result was if you wanted UNIX support you did it yourself or it didn't happen."[63]

Lions agrees, "We needed help," he notes, "but we couldn't get any from outside sources so we ended up generating our own expertise."[64] Not only did those working on Unix implementation at the University of New South Wales have access to the code, but Lions explains how Ian

Johnstone, the tutor working with him in his Operating Systems class, suggested making some of the code for the Unix kernel available to the students in his class. "I think it was in 1975," remembers Lions, that Ian Johnstone asked, "'Why don't we run off a few of the source files for the kernel and ask the students to take a look at them? Then we can ask them some questions; maybe it will be interesting'."[65] Lions took Johnstone's suggestion and made some of the Unix source code available to his class, but his students complained. They felt they needed to see the source code for the whole kernel in order to make sense out of any part.

Taking their suggestion, Lions decided to make a large part of the source code available to his class. "The following year," he recounts, "I prepared a booklet containing the source files for a version of Edition 6 UNIX that could then run on a PDP-11/40 system." Lions followed the book of source code with a book of "explanatory notes that was intended to introduce students to the code."[66] Lions explains that working on his book, "A Commentary on the UNIX Operating System" was a real learning experience. By slowly and methodically surveying the whole kernel he notes, "I came to understand things that others had overlooked."[67]

When he read the manual and wasn't quite certain about his interpretation, Lions would read the code. Through this process, he was able to determine that the manual was "really quite accurate in its description of what a program actually does. "In the Thompson/Ritchie era," he observes, "words were carefully chosen."[68] Lions writes, "In our opinion, it is highly beneficial for students to have the opportunity to study a working operating system in all its aspects."[69]

"Moreover," he adds, "it is undoubtedly good for students majoring in computer science to be confronted at least once in their careers with the task of reading and understanding a program of major dimensions."[70] Lions found that, "On the whole the authors of UNIX, Ken Thompson and Dennis Ritchie, have created a program of great strength, integrity and effectiveness," which he urged his students to "admire and seek to emulate."[71] Not only did students in Lions' class read and study the Unix source code and Lions' Commentary on the source code, but Lions describes how he sent more than 200 copies of his book to Bell Laboratories. Eventually, the Labs took over distribution of the book.

Tague relates how Lions' book of commentary and the Unix source code were used at AT&T "as part of the documentation package for those who wanted to understand or modify the UNIX® source code that the USG [Unix Support Group] shipped."[72] Even after Unix V6 had been replaced by Unix V7, Tague explains that Lions' Commentary continued to be useful as an introduction to Unix. Tague writes: "It outlined the conceptual architecture, very clearly in the short form of the system before it had accreted all the minor changes and feature additions that disguised the original clarity of its structure. All new people were given a copy when they joined the USG. And I suspect most development groups did the same."[73]

Pioneers like Henry Spencer describe how important it was to those in the Unix community to have the source code. He notes how having the sources made it possible to identify and fix the bugs that they discovered, "There is something the UNIX community has always been fairly strong on," he explained during an interview, "admitting things you know don't work about the software." Even in the late 1970's and early 1980's, remembers Spencer, "practically every UNIX site had complete sources."[74]

One of the early functions of Usenet, the early online community of Unix systems begun in 1979, according to Spencer, was to provide cooperative software support for the Unix community. He elaborates: "Well, for one thing, Usenet predated a lot of company bbs's and the like. It was basically a cheap way to hear about things fast and this was at a time when practically every UNIX site had complete sources and so a bug report often came with a fix. It was a way of finding out what people had discovered and what fixes they'd worked out for it. Quickly and easily. And for that matter, if you ran into something that you couldn't solve yourself, putting out an inquiry to a bunch of fairly bright people who were fairly familiar with the code, often got a response, 'O Yeah. We solved that one' or 'You're right. There's a bug. Here's how to fix it' or sympathy even if no one had a fix for it."[75]

Another Unix pioneer, Dick Haight, corroborates the important role open source code played for those in the Unix community: "That by the way, was one of the great things about UNIX in the early days: people actually shared each other's stuff. It's too bad that so many sites now have purposefully turned off the read privileges in order to keep their ideas from being stolen. Not only did we learn a lot in the old days from sharing material, but we also never had to worry about how things really worked because we always could go read the source. That's still the only thing that matters when the going gets tough."[76]

Unix was continually used and improved by its creators. A growing community of programmers, system administrators, and users, both at Bell Labs and at academic and research sites outside the Labs, also used, developed and debugged the code. The fact that the source code was open and available made this possible. The result was the creation of Unix as a powerful and flexible programming environment.

Though Unix was primarily designed to "help build research software," Bell Labs software developers soon found that, "what worked well in the programming laboratory also worked well on modest projects to develop minicomputer-based systems in support of telephone company operations."[77]

AT&T Automates Internal Operations

In the late 1960s, there had been an upsurge in demand for telephone company service which taxed all AT&T's resources.

The 1970 annual report of AT&T described the crisis: "The company concluded that the convergence of a number of adverse factors contributed to the crisis, including an unforecasted rapid upsurge of demand for business traffic on a poor switching system, faulty trunk maintenance and administration, inadequate training of maintenance personnel and poor main distributing frames administration."[78]

Section 1 of the U.S. Communications Act of 1934 charges the Federal Communications Commission (FCC) with promoting a "rapid, efficient, nationwide and worldwide wire and radio communications service with adequate facilities at reasonable charge."[79] This legislative obligation put pressure on AT&T to create and support the Bell Labs efforts to push forward the forefronts of computer science, since the most advanced technology and science were needed for AT&T to meet its obligations as a regulated communications utility.

Under pressure from both the New York Public Service Commission (NYPSC) and the FCC, AT&T had to find a way to upgrade their service.[80] In response to these requirements, a push began within AT&T to "use the computer to support the operation of the Bell System Network."[81] Berkley Tague, who would be instrumental in implementing this change at AT&T, describes the broad scope of the effort that AT&T instituted to deal with the service crisis. He lists: "Monitoring and alarms, maintenance and staff management, inventory and order control, revenue data collection and billing, traffic measurement and control, circuit provisioning,[82] etc." and he provides the following illustration: "The data that has to be collected to bill for calls was being stored on a variety of media – e.g. paper AMA tape, a punched paper tape a few inches wide, IBM compatible mag tape, and probably a few others. These tapes had to be mounted, saved, trucked to DP centers for processing, etc. They could be lost or damaged in the handling. An obvious solution was to send the data over data-links – our own network – to the DP centers. Minicomputer-based systems were designed to collect, preprocess and transmit this data in some cases; in others, the electronic switches themselves were programmed to do the job. The goal was to eliminate the need for people handling physical media – a process that is both expensive and error-prone."[83]

Software was needed to facilitate these changes. Tague explains how the Unix system, originally created for computer research, also served the needs of program developers. "The difference," he explains, distinguishing between the needs of researchers and the concerns of developers involved "support, stability and reliability." He describes how developers have to meet deadlines and so only tolerate absolutely necessary changes to the system. A researcher, on the other hand, according to Tague, will "tolerate a fair amount of upset and change and revision if it advances the toolkit significantly, even if it means you must recode your work."[84]

The use of Unix by developers for support operations began in the 1972 period. Tague describes how "a number of developers were planning to build their own operating systems – typically their first system and often their first major programs."[85] Having experience with Unix, he advocated Unix for the job. It was an uphill battle to convince developers to adopt Unix when there was no offered support and it was made up of 12,000 lines of undocumented code.

In 1973, Tague pointed out the need for a central support group and volunteered to form the group. He was given the job. The goal of the USG which Tague created in September 1973 was to "provide the stability and support that would buffer developers from the experiments that researchers were daily imposing on the evolving UNIX system." He describes how Dennis Ritchie promised him a portable version of Unix, which was delivered as promised by October, 1973.[86] Recoding the Unix kernel from assembler language, which was specific to each computer, into the C programming language, was an important feat. Once the Unix kernel was coded in C, it would become possible to implement Unix on different computers without having to rewrite the majority of the code.

The effort to make Unix machine independent was desirable, so that AT&T wouldn't be dependent on any one vendor for its computer hardware or software or service. He explains, "A goal of my effort was to remove the vendor lock so we could get competitive bids on the volume buys when we deployed these mini-based systems across the Bell System."[87]

Every project adopting Unix, Tague reports, added their own features and function to it and it became the responsibility of the USG to choose among or merge the variations. The University of California at Berkeley, according to Tague, performed a similar job with the variations of Unix developed by academic computer scientists outside of AT&T.

Creating a Programmer's Workbench

While Tague was working on Unix Support Operations, other Bell Labs researchers were planning another major project that would impact on software development and on the use of a new form of tool.

Rudd Canaday (who had designed the original Unix file system with Thompson and Ritchie), Dick Haight, Ted Dolotta, John Mashey, and others formed the Programmer's Workbench Group (PWB) within the Business Information Systems (BIS) project.[88] Tague explains that the "BIS problem was to get a common 'workbench' that would drive code onto any of the three or four commercial vendor's mainframes that was used by BIS." Tague writes that "By putting a UNIX® system in front of the large mainframe systems, developers got the advantages of UNIX® in developing code and a place they could store debugged standard command sequences that drove development and testing on the target mainframes."[89]

David Nowitz, a Unix pioneer who helped with the development of UUCP, and then went on to collaborate in the creation of honeydanber uucp, explains that when he hired into Bell Labs in 1969, there were programmers working on many different computers with different languages.[90] The problem these different computers represented for those at AT&T, is described by Haight: "We had a real problem to solve at the time. For one thing, we had a fairly large group of software developers at the Labs working on several different main frames. The biggest group, of course, consisted of people working on the IBM 360s or 370s. The programmers working on the IBM 360 or 370 had to contend with batch processing and IBM's Job Control Language (JCL) problems. Other programmers were working with the UNIVAC and another group with Xerox computers. Only a few were using early but expensive time sharing like TSO (IBM's timesharing) and Univacs Remand Service."[91]

"These systems," Haight relates, "not only offered very different programming environments, but proved to be very expensive to use and very unfriendly." Describing the task they were charged with, Haight continues, "Basically, we ended up trying to give all these people a cheap text-editing front end that they could use for interactive program entry. For that to happen, we needed to develop a programming environment that was consistent with what they already were using."

They developed a software subsystem which included five components that all of the programmers working on the different computers could use. The subsystem which they created was called the PWB. It included RJE (Remote Job Entry), an editor, a program for tracking version changes, a word processor and software to simulate test input.[92] Haight believed that Unix was "the best program development system around" and that this "development environment notion was the very concept that got PWB going back in '73."[93]

The vision of the PWB/UNIX was conceived of in mid April 1973 and installed on the first workbench machine in October 1973 on a PDP 11/45. By 1977 the PWB was serving 1000 users.[94] The programmers working on the PWB rejected the top down software design philosophy, i.e. of creating a fully integrated facility and implementing it, and then making it available to users. Instead they adopted a bottom up philosophy. They designed independent single function tools and made them available to users as early as possible. Users were encouraged to provide feedback. The PWB programmers had the actual feedback to help them determine what modifications were needed to improve the tools. Evan Ivie, a Bell Labs researcher who was one of the creators of the PWB, describes why the PWB/UNIX group adopted the bottom up philosophy of software design: "The idea of designing and building a complete and fully integrated workbench system was rejected for a number of reasons, not the least of which is the fact that no one knows what that system should look like at this point in time. Instead, every effort was made to identify some of the immediate needs of potential users and to develop pieces of the workbench that could satisfy those needs quickly. This approach provided the workbench designers with much valuable user feedback quickly, and it allowed the projects to start picking up pieces of the Workbench to satisfy their most critical needs immediately and thus to start a phased transition to the complete workbench."[95]

The design for implementing Unix on the IBM System/370 was done in 1979, coding was done in 1980, and by early 1981, the first production system IBM 3033AP was installed at Bell Labs at Indian Hill. Unix had been suggested as the development system for programmers for the switching system software for the 5ESS switch because the Programmer's Workbench provided "editors, source code control and software generation systems."[96]

In 1981, after a production Unix system was running on an IBM 3033AP in Indian Hill, the application software tools were ported from the PDP-11/70 computer to the 3033AP. Shell scripts were carried over and then users were transferred.[97]

By 1984 the largest user of the Unix system for the IBM System/370 was the development project for the 5ESS Switch. The creation at AT&T Bell Laboratories of software for the 5ESS switch required millions of lines of code.[98] The Unix operating system with the Programmer's Workbench software was chosen for the 5ESS switch software development project because the researchers felt that Unix "provided a better development environment than any other operating system available. In addition, the developers were all trained in using this system and all the software tools had been developed."[99]

The elaborate process of applying Unix to the IBM 3033AP is described by the researchers: "Over 300 tools, written in both C and shell command language, were identified and examined. After careful study, almost half of the tools were found to be little-used and were eliminated for porting to the 3033AP. The C programs required recompiling to generate objects that would run on a 3033AP; in general, they compiled without problems. The shell scripts were carried over with almost no problems. (After testing) System testing, which consisted primarily of generating the system software for the 5ESS switch was then done. The effort to port the application tools was small and again proved the strength and computer independence of the UNIX operating system and the associated application programs."[100]

The rationale leading to the creation of the Programmer's Workbench, is outlined by Ivie. He wrote: "Although the computer industry now has some 30 years of experience, the programming of computer-based systems persists in being a very difficult and costly job. This is particularly true of large and complex systems where scheduled slips, cost overruns, high bug rates, insufficient throughput maintenance difficulties, etc., all seem to be the rule instead of the exception. Part of the problem stems from the fact that programming is as yet very much a trial and error process."[101]

There are at this point, he observed, "only the beginnings of a methodology or discipline for designing, building and testing software. The situation is further aggravated by the rapidly changing hardware industry and by the continuing evolution of operating systems which continues to nullify much of the progress that is made in the development of programming tools. What can be done," he asked, "to move the programming industry toward a more professional and stable approach to software development?"[102]

After enumerating several of the alternatives, he explained that his proposal involved "a very different approach to improving the development process." His recommendation was "…that the programming community develop a program development 'faculty' (or facilities) much like those that have been developed for other professions (e.g. carpenter's workbench, dentist's office, engineer's laboratory)." Such an approach, he explained, "would help focus attention on the need for adequate tools and procedures; it would serve as a mechanism for integrating tools into a coordinated set; and it would tend to add stability to the programming environment by separating the tools from the product (the current approach is equivalent to carpenter's leaving their tools in each house they build)."[103]

The Business Information Systems Programs (BISP) area of Bell Labs which was to use the Workbench program had among the computers it used: an IBM 370/158 and 168, a Univac 1108 and 1110, two Xerox Sigma 5's and several minicomputers.[104] The Workbench encouraged the development of machine independent programming tools. "Each tool must now function," Ivie explained, "for programmers developing code for a number of different vendor machines. There is no room for shortcuts which are dependent on the idiosyncrasies of a given machine (or a given project.) One is thus forced into a more stable and generalized software development approach which should be more applicable to new machines."[105]

Summing up the rationale of the Programmer's Workbench philosophy, Ivie wrote that "the programming profession has yet to produce a software development methodology that is sufficiently general so that it can be transferred from one project to another and from one machine to another." With the development of a Programmer's Workbench, he hoped, "a machine dedicated to the software development and maintenance function can serve as a vehicle for the development of such a methodology."[106]

Rationalizing the Process of Software Production

In their article published in *Scientific American* in 1966, Fano and Corbato noted that the creation of a working computer in 1944 was followed by a series of quantum leaps in both the speed of the electronics and the development of special languages which facilitated human-computer communication. In another article in the same special issue of *Scientific American*, John McCarthy identified the importance of being able to program for making use of the computer. He also noted that spending time mastering a programming language was less important than understanding the functions the software is to serve. Their descriptions demonstrate that the computer is an all-purpose machine that can be directed to do whatever tasks humans can conceive of programs to describe. This poses a dilemma. Since humans are still needed to create the directives for the computer, the challenge raised is how can the process of programming be made more rational and productive.

Perlis had observed that the technical progress of a society was marked by the development of tools and the ever developing set of people capable of wielding those tools. He noted the continuing evolution of the process of producing tools. In the evolution of Unix, the need for software tools became evident and a series of tools was developed to meet these needs. The creation of time-sharing and the development of Unix as a time-sharing system has helped to create a community of people who not only use Unix, but who also continue its development. The contributions to the evolution of Unix by researchers at Bell Labs and then by those at the University of California at Berkeley, and other users and programmers around the world who continued the development of Unix, are an important achievement in the development of the process of software creation.

Writing in 1975, one of the team of researchers creating the PWB, P. J. Plauger, observed that "building on the work of others is the only way to make substantial progress in any field."[107] Describing the process of creating software tools, Plauger explained that each tool is a small programming project requiring anywhere from a few man months to a few hours to complete. He proposed that these tools need to be designed in a standard way so that they can interface with and be combined in a useful way with other tools for a variety of different purposes. "The programmer working this way becomes a tool builder, and his impact extends far beyond the task he may originally have set out to solve," he concluded.[108]

Placing the creation of software tools into the broader context of tool production provides a basis to evaluate the achievements and problems of this new stage. Sigvard Strandh, in the History of the Machine, describes how mechanical hand tools had acquired a stable form at an early stage. "From the Middle Ages until the era of industrialism," he writes, "the tools of the carpenter, joiner and smith remained more or less the same." With the publication of the Great French Encyclopedia (between 1751 and 1780), by Diderot and d'Alembert, however, Strandh observes that the development of these tools was given a substantial boost. Publication of the descriptions of the tools used in all the trades extent at the time, in great detail, in the Great French Encyclopedia, made it possible for those interested to learn how to build the tools, "so as to establish himself in one of the trades described."[109]

The publication in the Great French Encyclopedia made possible the constant improvement of the old hand tools, and thus made it possible to manufacture the new tools needed for the new technical developments. "Industrialization," Strandh writes, "means the manual methods of manufacture are mechanized, that is, taken over by machines. The machine tool is the local outcome of the process. Mass manufacture of machine tools," which he writes, "has been called 'the industry of industries', didn't become widespread until the 1870's."[110]

The conditions described by Strandh made possible advances in the technology of tool production in the 19th Century. These conditions provide a helpful perspective when examining the achievements of the Unix community in developing the software tool creation process. Bell Labs researchers early on recognized that software production required text editing in place of soldering, and that this substitution represented a significant challenge. The creation by Bell Labs programmers of software tools, making it possible to produce better software more efficiently, is an important development. This development is similar to the ability to improve hand tools that Morgan documents as the technological advance represented by the invention of iron. The success of writing the code for the 5ESS switch at AT&T demonstrates the achievement these software tools have made possible.[111]

Generating software is different from creating hardware and it presents a particular challenge. You can't see software tools or touch them or handle them as you can mechanical tools. "Software is invisible and unvisualizable," Brooks points out, since "the reality of software is not inherently embedded in space." Therefore, he observes, "In spite of the progress made in restricting and simplifying the structures of software, these structures remain inherently unvisualizable and thus do not permit the mind to use some of its most powerful conceptual tools. This lack not only impedes the process of design with one mind, it also severely hinders communication among minds."[112]

Therefore, there is a particular problem posed when creating, describing, or representing software tools. Diderot was able to catalog the tools and industrial processes of his times, but such a task is much more difficult when it comes to creating such an encyclopedia or presentation of software tools, or in creating the software component catalog that McIlroy once proposed. The development of UUCP and then of Usenet, as a means of cheap communication among those of the Unix community, made it possible to surmount some of the difficulties inherent in the process of software production.[113]

The Usenet community was able to pioneer the creation of a cooperative network that helped facilitate communications among the Unix software community. "Though large institutions have been able to avail themselves of communications networks such as ARPANET, the UNIX community has made inexpensive electronic communication available to all its members via Usenet," remarked John Stoneback, a faculty member at Moravian College, in an article describing the important role that Unix played for members of the academic community.[114]

Describing the importance of the creation of the Unix communication program UUCP, McIlroy writes that "before uucp remote login was the only kind of electronic connection between Unix systems. With uucp, every PDP 11 with dial-up (not even dial-out) access became connected. E-mail and file transfer previously available only among the fortunate few sites on ARPANET, were democratized overnight. This was the brainchild of Mike Lesk."[115] Lesk's original plan in creating UUCP was to create an automatic software updating system.[116] That, however, was fraught with difficulties, and instead, with modifications to UUCP by David Nowitz, it became the basis for a broad ranging online computer network that came to be called Usenet. McIlroy, describing how uucp was used by Tom Truscott and other Usenet pioneers to create an online community, writes, "Soon Truscott saw that the new medium could be used for automatic dissemination of information, and netnews was born."[117]

The Unix system is based on a methodology that helps the user to learn the new language. The methodology is similar to the way children learn to talk. As Unix pioneers Brian Kernighan and John Mashey, write, "The code that people see, adapt, and imitate is usually well structured. People learn to code well in the same way that they learn to speak their native language well by imitation and immediate feedback."[118]

"People often look at each other's code," they explain, "comment on it in person and through interuser communication facilities, and take pieces of it for their own use. The ideas of programming teams and egoless programming fit into the Unix environment well, since they encourage sharing rather than isolation."[119] Given the frequent sharing and building on each others code, though, "some programs have always been 'owned' by one or two people, many others," however, they note, "have been passed around so much that it is difficult to tell exactly who wrote them…." Also, they point out that "Tools have been built to automate many common programming tasks."[120]

Kernighan and Mashey explain how in its early days, up through 1974, "Unix best supported a single, cooperative, tightly coupled group of people on each machine." By 1975, the PWB/UNIX made possible support for a larger number of users. They describe how there have been large groups of over 200 users working on different machines, but that users preferred "to share a machine so they could share procedures and data bases."[121] They describe how the concept of software tools spread from the Unix Programming environment and PWB/UNIX to more generalized applications described by Kernighan and Plauger in their book Software Tools (1976), then to further applications that were ported to other machines, and then to the more generalized applications of the Lawrence Berkeley Laboratory (LBL).[122]

Documenting Technological Progress

In writing about the role that technology plays in the development of society, Morgan observed that oral or written documentation helps to facilitate technological progress. Morgan described how the Greek poet Homer by documenting the civilization that existed prior to his times, provided future generations with a yardstick by which they could measure their own accomplishments.[123]

So, too, documentation of the achievements of the Unix pioneers and of the tools they created, is an important step to be taken by those wanting to understand the progress made in our times. When amateur scientists created the Royal Society in Great Britain in the 1660's, one of their early efforts was to invite Rev. Spratt into the Society and to support him to write a history of the documentation of technology from Homer to their own times. Diderot's work on the Great French Encyclopedia, in the years leading up to the French Revolution, played such a role summarizing the knowledge, including, most importantly, the summary of all the trades and industries with their tools existing up to his time in France.

Unix pioneers often emphasize that scientific progress can only be made by building on what has already been achieved. It is important, therefore, to recognize the legacy of how Unix made it possible to achieve programming projects involving millions of lines of code by creating a workbench of software development tools. The creation of a new form of tool requires a new form of tool production process and though such technology is only in its infancy, there are examples to be studied to know what we are building on. As Henry Spencer and Geoff Collyer recognized when they were working on increasing the performance of Netnews software, it was necessary to systematically record what was happening to be able to see what progress was being made, "To know how to get somewhere, you must know where you are starting from."[124] Similarly, it is important to systemmatically document the history of the significant developments of the software tool revolution, so that there will be a yardstick by which to measure the accomplishments and to determine the new questions to be explored.[125]

Notes for Chapter 9

1. Lewis Henry Morgan, Ancient Society, Chicago, 1877, p. 42.

2. Alan Perlis, Introduction to Computer Science, New York, 1972, p. 4.

3. Ibid., p. 3.

4. Ibid.

5. See F. J. Fano and R. M. Corbato, "Time Sharing on Computers" in Information, San Francisco, 1966, p. 77. See also C. Strachey, "Time Sharing in Large Fast Computers," Proc Int Conf on Info Processing, UNESCO, June, 1959, pp. 336-341. See also Frederick P. Brooks Jr., The Mythical Man-Month, Reading, 1972, p. 146. Brooks describes how both E. F. Codd and C. Strachey reported work aimed at time-sharing debugging in 1959. According to Brooks, Codd's system, which emphasized more efficient input-output and interactive debugging was not actually implemented while Strachey's work was built on by Corbato and others at MIT.

6. Early time-sharing systems were developed at Bolt, Beranek and Newman (BBN), a company then specializing in acoustical research, at MIT's Research Laboratory of Electronics (RLE) and at System Development Corporation (SDC).

See Fano and Corbato, Ibid., p. 78.

7. Ibid.

8. Ibid., p. 94. They go on to explain: “Indeed, the analogy is not farfetched; an editorial board representing the community of users acts as a referee to pass on all new commands that are to be introduced into the system and on all information that is to be stored in the public files.”

9. Ibid., p. 95.

10. Ibid.

11. John McCarthy, “Information” in Information, San Francisco, 1966, p. 1.

12. Ibid., p. 4.

13. Ibid., p. 8.

14. Ibid., pp. 15-16.

15. Doug McIlroy notes that, “The Multics planners already had seen remote access protyped at several places, including CTSS and some commercial vendors. Their bigger vision was of an ‘information utility,’ not just a remote-access system that allowed economies of scale in computer cycles.” From e-mail correspondence July 30, 1995. Documenting the need for new hardware as well as software, E. L. Glaser, J. F. Couleur and G. A. Oliver, in “System Design of a Computer for Time Sharing Applications,” explain: “Investigation proved computers that were immediately available could not be adapted readily to meet the difficult set of requirements time-sharing places on any machine. However, there was one system that appeared to be extendible into what was desired. This machine was the General Electric 635.” (Proc-FJCC, AFIPS, vol 27, part 1, November 30, 1965, p. 197.)

16. In their article, “Introduction and Overview of the Multics System,” F. J. Corbato and V. A. Vyssotsky explain: “For several years a solution has been proposed to the access problem. This solution usually called time-sharing, is basically the rapid time-division multiplexing of a central processor unit among the jobs of several users, each of which is on-line at a typewriter-like console.” (Ibib., p. 186)

17. Ibid., p. 185.

18. “Automating Telephone Support Operations: An Interview with Berkley Tague,” *Amateur Computerist*, vol. 6 no. 1, Winter/Spring, 1994, p. 8.

19. Ibid.

20. “Interview with Victor Vyssotsky,” Unix Review, January, 1985, p. 59.

21. Ibid.

22. Ibid.

23. Ibid., p. 60.

24. Dennis M. Ritchie, “The Evolution of the UNIX Time-sharing System, AT&T Bell Labs Technical Journal, vol. 63 no 8, part 2, October, 1984, p. 1578. (Hereafter, “Evolution”)

25. Dennis Ritchie, “The Development of the C Language”, ACM, presented at the Second History of Programming Languages Conference, Cambridge, Massachusetts, April 1993, p. 1.

26. “Interview with Victor Vyssotsky”, p. 60.

27. D. M. Ritchie, “UNIX: A Retrospective”, The Bell System Technical Journal, vol. 57 no. 6, part 2, July-August, 1978, p. 1950. Ritchie refers to the article by R. J. Feirtag and E. I. Organick, “The Multics input-output system,” Proc. Third Symposium on Operating Systems Principles (October 18-20, 1971), pp. 35-41.

28. “Evolution,” p. 1580.

29. “The Development of the C Language,” p. 2.

30. Ibid.

31. Ibid.

32. “Evolution,” p. 1580. Also, there is a interesting account by Ritchie of the difference between the work at Bell Labs with Unix as opposed to their work on Multics. (See “Evolution,” p. 1587.)

33. E-mail correspondence from Doug McIlroy, July 30, 1995.

34. The BCPL text formatter had been inspired by J. Saltzer’s runoff program for CTSS.

35. “Evolution,” pp.1589-90. In the lobby of the main building at Bell Labs in Murray Hill, N.J. there is an exhibit of communication breakthroughs which credits Doug McIlroy and Rudd Canaday with the invention of pipes.

36. E-mail exchange with Doug McIlroy. When asked why he had proposed that pipes be introduced into Unix, McIlroy responded: “Data streams were a particularly clean notion of program composition that I’d talked about for a long time. I proposed them in an unclarified way for Multics in 1964. And Joe Ossanna actually wrote some elaborate IO switching for Multics, though I don’t think it ever caught on. I had been much taken by Conway’s idea of coroutines, which was published (in CACM?) around 1963, and was brought to high development by Dahl in Simula. I thought a lot about stream representations of algorithms. Indeed, the algorithm I am proudest of is the coroutine sieve of Eratosthenes that I described at Cambridge in 1968, and which is now one of the chestnuts of parallel computing.” (from e-mail communication)

37. “Evolution,” p. 1590.

38. “Interview with Victor Vyssotsky,” Unix Review, January, 1985. “Co-routines had after all shown up in SIMULA by the end of 1967.”

39. In August Mohr, “The Genesis Story,” Unix Review, January, 1985, p. 5. Mohr explains, “Haight believes that the pipe facility has had a major effect on the evolution of UNIX. Because of the shell and the piping facility, programs running under UNIX can be conversational without being explicitly programmed that way. This is because piping adds to the shell’s ability to accomplish this purpose for the system as a whole.”

40. M. D. McIlroy, “Unix on My Mind”, Proc. Virginia Computer Users Conference (Blacksburgh), vol. 21, September, 1991, pp. 1-6.

41. See M.D. McIlroy, “A Research UNIX Reader: Annotated Excerpts from the Programmer’s Manual, 1971-1986". Computing Science Technical Report, No. 139, AT&T Bell Laboratories, Murray Hill, New Jersey, June 1987, pp. 9 and 43.

42. Account provided by Doug McIlroy

43. “A Research UNIX Reader,” p. 24.

44. Account provided by Doug McIlroy.

45. See for example Frederick Brooks, “No Silver Bullets,” Unix Review, November, 1987, p. 41 and also The Mythical Man-Month, Addison-Wesley, Reading, MA, 1982, for a helpful statement of the problem in programming research that needed to be solved.

46. “Unix on My Mind.”

47. E-mail correspondence from Doug McIlroy.

48. The Unix Programming Environment, Englewood Cliffs, New Jersey, 1984.

49. Brian Kernighan and Rob Pike, “Program Design in the UNIX Environment,” in AT&T Bell Labs Technical Journal, October, 1984, p 1601.

50. Ibid., p. 1596.

51. M.D. McIlroy, E.N. Pinson, and B. A. Tague, “Foreword”, Bell System Technical Journal, vol. 57 no. 6, part 2, July-August, 1978, p. 1901.

52. Ibid., p. 1899. See for example, the thunderclap in the Appendix.

53. Ibid., p. 1902. See also Stephen C. Johnson, “UNIX: The Language Forms”, in USENIX Association Winter Conference Proceedings, Washington, D.C., January 21-23, 1987, p. 18.

54. See Brian W.Kernighan and Rob Pike, The Unix Programming Environment, Englewood Cliffs, New Jersey, 1984, p. 200. “idiff is loosely based on a program originally written by Joe Maranzano. diff itself is by Doug McIlroy, and is based on an algorithm invented independently by Harold Stone and by Wayne Hunt and Tom Szymanski. (See “A fast algorithm for computing longest common subsequences,” by J. W. Hunt and T. G. Szymanski, CACM, May, 1977.) The diff algorithm is described in M. D. McIlroy and J. W. Hunt, Technical Report 41, 1976. To quote McIlroy, “I had tried at least three completely different algorithms before the final one. diff is a quintessential case of not settling for mere competency in a program but revising until it was right.”

55. “Foreword”, p. 1902.

56. D. M. Ritchie and K. Thompson, “The UNIX Time-Sharing System,” The Bell System Technical Journal, vol. 57 no. 6, part 2, July-August, 1978, p. 1926.

57. Fred P. Brooks, Jr., “No Silver Bullets”, UNIX Review, November, 1987, p. 41.

58. Ibid.

59. Michael Lesk, “Can UNIX Survive Secret Source Code?”, Computing Systems, vol. 1 no. 2, Spring, 1988, p. 189.

60. D. M. Ritchie and K. Thompson, “The UNIX Time-Sharing System,” p. 1927.

61. Marshall Kirk McKusick, “A Berkeley Odyssey”, Unix Review, January 1985, p 30.

62. See Peter Ivanov, "Interview with John Lions," UNIX Review, October, 1985, p. 51.

63. "Interview with Henry Spencer: On Usenet News and C News," The *Amateur Computerist*, vol. 5 no. 1-2, Winter/Spring 1993, p. 5.

64. "Interview with John Lions," p. 51.

65. Ibid., p. 52.

66. Ibid.

67. Ibid.

68. Ibid., p. 53.

69. John Lions, A Commentary on the UNIX Operating System, The University of New South Wales, (nd), p. 5.

70. Ibid.

71. Ibid., p. 9. A short time later, in "Unix Implementation", The Bell System Technical Journal, vol. 57, no. 6, part 2, July-August 1978, Ken Thompson described how a goal in writing the Unix kernel was to write exemplary source code that others would emulate. Thompson writes: "What is or is not implemented in the kernel represents both a great responsibility and a great power. It is a soap-box platform on 'the way things should be done.' Even so, if 'the way' is too radical, no one will follow it. Every important decision was weighed carefully. Throughout, simplicity has been substituted for efficiency. Complex algorithms are used only if their complexity can be localized."(pp. 1931-32)

72. "Automating Telephone Support Operations: An Interview with Berkley Tague," *Amateur Computerist*, vol. 6 no. 1, p. 10.

73. Ibid. Also, Mike Blake-Knox describes how he did development work for Bell-Northern Research (BNR), which was the Bell Labs equivalent for Northern Electric (which is now Northern Telecom). He reports that he had a copy of Lions' book and used it to do Unix kernel modifications. (from e-mail correspondence Feb. 13, 1994)

74. "Interview with Henry Spencer," p. 4.

75. Ibid.

76. Marc Rochkind, "Interview with Dick Haight," Unix Review, May, 1986, p. 65.

77. M. D. McIlroy, E. N. Pinson, and B. A. Tague, "Foreword", Bell System Technical Journal, vol. 57, no. 6, July-August, 1978, part 2, p. 1902.

78. Alan Stone, Wrong Number, New York, 1989, p. 17.

79. Ibid., p. 145. "For example, since 1893, Bell maintenance employees had left unneeded jumper wires attached to main distributing frames [which connect each customer wire pair to the line terminal in a switching center] with no ill effect, but the growth in traffic caused the weight of dead jumpers to impede the operation of the main distributing frames."

80. During this period the NYPSC imposed standards requiring regular reporting from AT&T so that progress in upgrading its service could be monitored. Also in 1970 the FCC required monthly reports on the quality of AT&T service in 20 major cities and by 1972 in 72 administrative areas.

81. Tague situates this push as beginning around 1969.

82. "Interview with Berkley Tague," pp. 7-8.

83. Ibid., p. 8.

84. Ibid., p. 9.

85. Ibid.

86. Ibid. McIlroy in e-mail correspondence clarifies, "The conversion of Unix to C in 1973 was a step toward portability, but was certainly far from achieving it. It took more work to port Unix to the 360 than to make Unix in the first place. Not until 1978 did Richard Miller in Australia and Ritchie and Johnson at AT&T demonstrate effective portability."

87. See S. C. Johnson and D. M. Ritchie, "Portability of C Programs and the UNIX System," The Bell System Technical Journal, vol. 57 no. 6, part 2, July-August 1978, pp. 2021-2048.

88. The Programmer's Workbench was first suggested by E. L. Ivie. Ivie, E. L. "The Programmer's Workbench – A Machine for Software Development", Unpublished Report, Bell Laboratories, May 19, 1975.

89. "Automating Telephone Support Operations: An Interview with Berkley Tague, The *Amateur Computerist*, vol. 6 no. 1, p. 10.

90. Uucp was first designed and implemented by Mike Lesk of Bell Labs in 1976. It has undergone many revisions and improvements. It was further developed by Lesk and David Nowitz and made available outside of the Labs as part of the V7 distribution. A new revision and improvement known as the Honeydanber UUCP was written in 1983 by Peter Honeyman, David A. Nowitz and Brian E. Redman.

91. "Interview with Dick Haight," p. 55.

92. The subsystems that became the PWB included: 1) the ed line editor, 2) a modified RJE, 3) a piece to simulate test input to applications for the IBM and Univac and Xerox computers, 4) SCCS software tool for tracking program changes, and 5) the mm macros so that people could use the system for word processing.

93. "Interview with Dick Haight," p. 63.

94. See "The Programmer's Workbench" by T. A. Dolotta and R. C. Haight and J. R. Masey, in Proc. Second Int. Conf. on Software Engineering, October, 13-15, 1976, pp. 2180-2181.

95. Evan L. Ivie, "The Programmer's Workbench – A Machine for Software Development" Communications of the ACM, vol 20 no 10, October, 1977, pp. 750-751. Ivie's paper described the implementation of 5 basic components of the workbench: job submission, module control, change management, document production and test drivers.

96. W.A. Felton, G.L. Miller and J. M. Milner, "A Unix System Implementation for System/370," AT&T Bell Labs Technical Journal, October, 1984, p. 1752.

97. Ibid., p. 1753.

98. Ian Johnstone, who had been the tutor at University of New South Wales working with Professor John Lions, was one of the researchers invited to Bell Labs. He managed the completion at AT&T Bell Labs of the port of Unix to the IBM 370 computer. See "Unix on Big Iron" by Ian Johnstone and Steve Rosenthal, UNIX Review, October, 1984, p. 26. Johnstone also led the group that did the port to the AT&T 2B20A multiprocessor system.

99. “A Unix System Implementation,” p. 1753.

100. Ibid., pp. 1763-4.

101. Evan L. Ivie, “The Programmer’s Workbench – A machine for Software Development”, Communications of the ACM, vol 20 no 10, October, 1977, p. 746.

102. Ibid.

103. Ibid.

104. Ibid., p. 747.

105. Ibid., p. 749.

106. Ibid.

107. P. J. Plauger, AFIPS Conf Proceedings, vol 44, 1975, p. 281.

108. Ibid.

109. Sigvard Standh, The History of the Machine, New York, 1989, p. 79.

110. Ibid., p. 81.

111. See for example, articles in the AT&T Bell Labs Technical Journal, July-August, 1985, vol. 64 no. 6, part 2 on “The 5ESS Switching System” and “A UNIX System Implementation for System/370" by W. A. Felton, G. L. Miller, and J. M. Milner, in Bell Laboratories Technical Journal, vol. 63 no. 8, part 2, October, 1984, p. 1751.

112. Fred P. Brooks, Jr., “No Silver Bullets”, The Unix Review, November, 1987, p. 41.

113. See for example “On the Early days of Usenet,” Chapter 10 of this book.

114. John Stoneback, “The Collegiate Community,” Unix Review, October, 1985, p. 26.

115. Doug McIlroy, e-mail correspondence.

116. Mike Lesk, “A Proposal for Operating System Support.”

117. McIlroy correspondence.

118. Brian W. Kernighan and John R. Mashey, “The Unix Programming Environment,” Computer, April, 1981, p. 20.

119. Ibid.

120. Ibid.

121. Ibid.

122. Like Burroughs B1726. See C. R. Snow, “The Software Tools Project” in Software - Practice and Experience, vol. 8, September-October, 1978, pp. 585-99. See also “A Virtual Operating System” by Dennis E. Hall, Deborah K. Scherrer, and Joseph S. Sventek, in Communications of the ACM, vol. 23 no. 9, September, 1980, pp. 495-502.

123. Morgan writes: “The use of writing or its equivalent in hieroglyphics upon stone, affords a fair test of the commencement of civilization. Without literary records neither history nor civilization can properly be said the exist. The production of the Homeric poems, whether transmitted orally or committed to writing at the time, fixes with sufficient nearness the introduction of civilization among the Greeks. These poems, ever fresh and ever marvelous, possess an ethnological value which enhances immensely their other excellences. This is especially true of the Iliad, which contains the oldest as well as the most circumstantial account now existing of the progress of mankind up to the time of its composition.” (Ancient Society, p. 31)

124. “Collyer kept detailed records of his work on ‘rnews’, so he could see how much progress he was making.” Geoff Collyer and Henry Spencer, “News Need Not Be Slow,” Winter 1987 USENIX Technical Conference Proceedings, Washington, D.C., p. 183.

125. Though there is a rich technical literature documenting Unix development in various technical journals, historical accounts are harder to find and often hard to acquire extent copies. Some of the helpful sources include “Unix Time-Sharing System” Special issue of The Bell System Technical Journal, vol. 57 no. 6, Part 2, July-August, 1978, and “The Unix System” special issue of AT&T Bell Laboratories Technical Journal, vol. 63 no. 8, part 2, October, 1984. Also, Life with Unix by Don Libes and Sandy Ressler, Prentice Hall, Englewood Cliffs, N.J., 1989. Numerous interviews and articles with or about Unix pioneers appeared in early issues of Unix Review during the 1980's. Also, the issue of The *Amateur Computerist* celebrating the 25th Anniversary of Unix, vol. 6 no. 1, Winter/Spring 1994 is available online. The URL is http://www.columbia.edu/~hauben/acn/ and contains interviews with Unix pioneers John Lions and Berkley Tague, as well as an article on the development of Unix. The book A Quarter Century of UNIX, by Peter Salus, Addison-Wesley, 1994, contains references to a set of interviews done by Mike Mahoney from “The Unix Oral History Project: Release.0, The Beginning,” ed. and transcribed by Michael Mahoney as well as other interviews.

Thanks to Doug McIlroy, David Nowitz, John Lions, and Berkley Tague, for making available information that was helpful in doing the research for this article and thanks to Doug McIlroy, Alfred Aho, Mark Horton and Tom Truscott for commenting on earlier versions.

Appendix

“Like a thunderclap from the sky, the pipeline slays the problem.”

Following is a script provided by Tom Truscott. Such particularly powerful pipelines have been referred to by Ken Thompson as “thunderclaps”.****

To find how often each word appears in files named book.1, book.2, ...:

```
cat book* | tr -cs ‘[A-Z][a-z]’ ‘\012' | sort | uniq -c | sort -nr | more
                                ------
```

Truscott notes “Alas, thanks to so many *IX variants this pipeline doesn’t always work, on some the underlined text needs to be ‘[\012*]’.”
